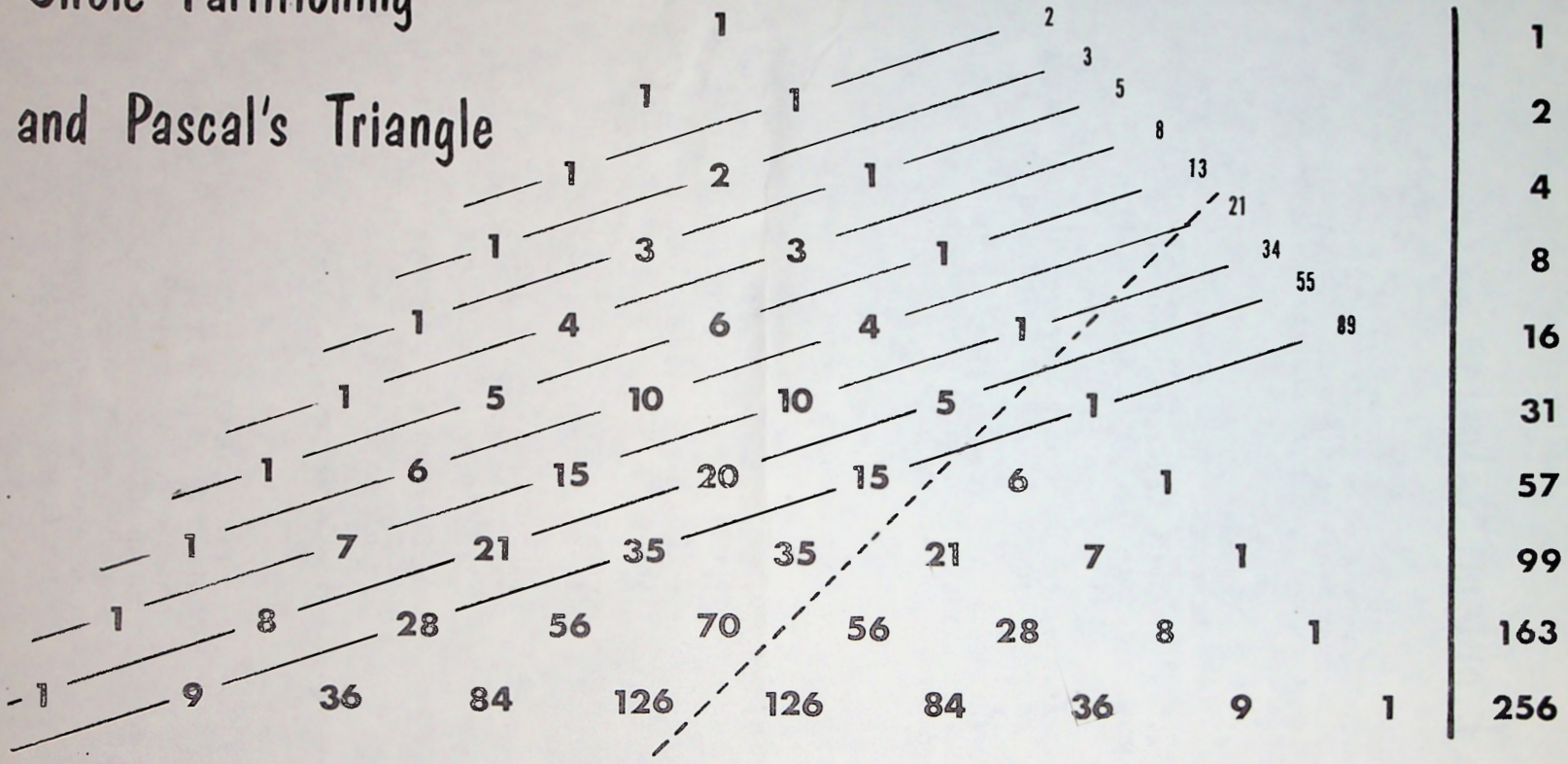
Circle Partitioning ~~ and Pascal's Triangle



Popular Computing

Volume 8
Number 3

The world's only magazine devoted to the art of computing.

March 1980

84

Problem Solution

SWITCH HAPPY (Problem 265, issue 81) presented a flowchart with four programmed switches. The entire action of the flowchart dealt with manipulating those switches, and counting the number of traverses of the 12 arms of the flowchart.

Specifically, the problem asked for the relative number of times that the complex path would reach Reference number 7. The intent of the problem was to make vivid the concept that the result of executing a computer program is highly unpredictable, especially by the program's author.

Bob Hall, Arleta, California, programmed the actions called for in the flowchart on his HP-97 calculator and reports that Reference 7 is never reached. On each return to Reference 2, the switches are set to sides 3, 3, 3, and 1 respectively, and switch D is never set to its side 3 when passing Reference 5.

Mr. Hall concludes:

1. Writing such a program is simple.
2. Analyzing the operation of this system (or its flowchart) without running the program is tedious, but could be done.
3. For very large systems, human analysis without computer assistance would be virtually impossible. Preventing recording errors alone would be an extremely formidable task.
4. Reliable analyses of large systems of these types could only be performed by computer.
5. Since reliability is necessary in such analyses, computers are the only feasible analytical tools.

Publisher: **Audrey Gruenberger**
Editor: **Fred Gruenberger**
Associate Editors: **David Babcock**
Irwin Greenwald
Patrick Hall
Contributing Editors: **Richard Andree**
William C. McGee
Thomas R. Parkin
Edward Ryan
Art Director: **John G. Scott**
Business Manager: **Ben Moore**

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 per year. For all other countries, add $3.50 per year. Back issues $2.50 each. Copyright 1980 by POPULAR COMPUTING.

Problem Solution

Circle Partitioning

BY THOMAS R. PARKIN

The problem that was posed in issue number 36 was this: to divide the circumference of a circle into 1, 2, 3, 4,... (not equal) parts, connect all pairs of points, and determine into how many pieces the circle is then divided. From the figures that appeared with the problem, we have this data:

N	Number of pieces
1	1
2	2
3	4
4	8
5	16

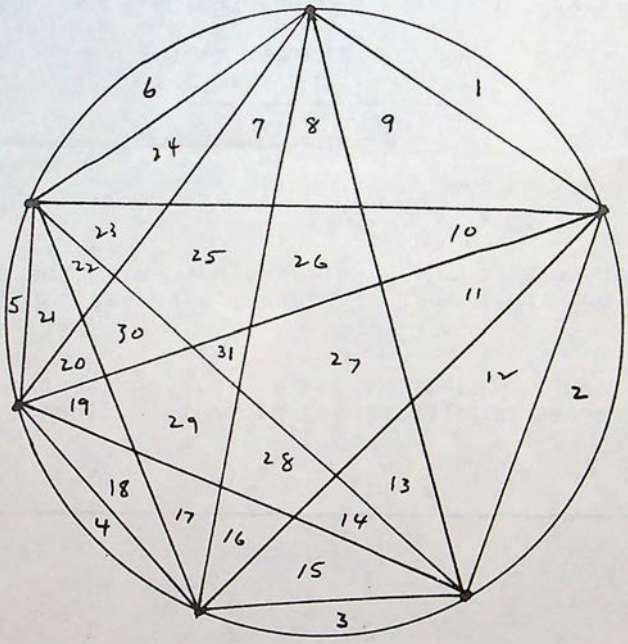
(Figures for cases 6 and 7 were included with the original statement, but the number of pieces is not so readily seen.)

When confronted with such a table, one method of approach is to form a set of differences along this line:

N	f(N)	Δ_1	Δ_2	Δ_3	Δ_4
1	1				
		1			
2	2		1		
		2		1	
3	4		2		1
		4		2	
4	8		4		
		8			
5	16				

Unfortunately, in this case, the process doesn't yield enough data to allow extending the table, yet. None of the differences (as far as we can tell) is a constant from one level to the next (down the columns).

The next thing one might try is analysis of the basic operation, noting things like: for n points there are n connecting lines which cut no other lines; the first of n points yields (n - 1) lines to other points, the 2nd yields (n - 2), and so on; there are n pieces formed by the first (n - 1) lines between those lines and the circumference... and so forth. Such an attack frequently yields an insight which allows one to guess the answer which can then be proved. However, before proceeding along this line very far, we realize that at some point we shall probably have to draw at least one more case beyond N = 5 to check our work. (Obviously the first guess that for N = 6, f(N) = 32 is just a bit too easy!) Well, let's draw the figure for 6 unequal points, and number the pieces in some systematic way:



Thus we see that for N = 6, f(N) = 31, rather a surprise. Now let's see what our table of differences tells us:

N	f(N)	Δ_1	Δ_2	Δ_3	Δ_4
1	1				
		1			
2	2		1		
		2		1	
3	4		2		1
		4		2	
4	8		4		1
		8		3	
5	16		7		
		15			
6	31				

We now see that the 4th differences are constant and we can immediately write down a 4th degree polynomial which will yield f(N) for any value of N. In order to do this, we invoke the Gregory-Newton formula for interpolation:

$$y = f(a) + x\,\Delta f(a) + \frac{x(x-1)}{2!}\,\Delta^2 f(a)$$

$$+ \frac{x(x-1)(x-2)}{3!}\,\Delta_3 f(a)$$

$$\ldots + \frac{x(x-1)\ldots(x-n+1)}{n!}\,\Delta_n f(a);$$

where f(a) is the functional value for x = a, $\Delta f(a)$ is the first difference of f(x) at $a+\omega$, $\Delta_2 f(a)$ is the second difference at $a+2\omega$, etc., and ω is the interval between successive x's, in this case 1.

For f(a) we can take the value at N = 1, f(N) = 1. This then gives us

$$\Delta f(a) = \Delta_2 f(a) = \Delta_3 f(a) = \Delta_4 f(a) = 1,$$

all from the top entries of the columns of the difference table. Now, we write down the polynomial, reduce, expand, and collect terms:

$$f(N) = 1 + N\cdot 1 + (1/2!)\cdot N\cdot(N-1) + (1/3!)\cdot N(N-1)(N-2)\cdot 1$$

$$+ (1/4!)\cdot N(N-1)(N-2)(N-3)\cdot 1$$

$$f(N) = (N^4 - 2N^3 + 11N^2 + 14N + 24)/24.$$

We can substitute a few values for N in this formula and see how the series goes. Also, of course, using the constant 4th differences, we can extend our difference table back to the functional value as far as we like. Then, as a further check, we can draw the figures for N = 7 and beyond and actually count the spaces.

All these techniques should, and will, yield the same series, which goes like this:

N	f(N)
1	1
2	2
3	4
4	8
5	16
6	31
7	57
8	99
9	163
10	256
11	386
12	562
13	794

and we are now well beyond drawing the figures!

It would be possible to continue the analytic approach we started with in our earlier observations and deduce the formula we obtained from the difference table. One can easily see the development of the terms of the unexpanded polynomial by drawing a few figures and observing the number of lines which cross other lines and how this changes with N. Furthermore, the terms for N and 1 are both quite apparent (from the periphery and the center). However, let's explore another avenue altogether.

We note from the unexpanded form of the interpolation polynomial that the terms appear to contain factorials and partial factorials the likes of which should be familiar to us from the binomial theorem. Furthermore, we know that the sums of the coefficients in binomial expansions are powers of two and we seem to have a few of those. Let's write out the elements generated by the binomial expansions in a form known as Pascal's Triangle--on the cover of this issue.

It is, of course, obvious from whence all those interesting series of numbers come. Perhaps another time we can explore this fascinating topic of basic combinatorial theory.

Now, isn't that interesting? We have related a simple puzzle to Newton-Gregory, Pascal, and the binomial theorem, known since Euclid!

Case Study 1

The PAYDAY Problem

What beginners at computing need is practice in carrying a problem situation through all its stages, from conception to correct output. Observation of some case studies might help. Try this one.

The problem statement is this: An industrial firm in the United States pays its employees twice a month, according to the following schedule:

1. Pay checks will normally be issued on the 5th and 20th of each month, if those dates are working days.

2. If the 5th and 20th fall on weekends (i.e., Saturdays or Sundays), payment will be made on the preceding Friday.

3. If the 5th or the 20th fall on a legal holiday, other than a Saturday or Sunday, payment will be made on the next following working day. Legal holidays for this company are defined to be:

New Year's	January 1
Christmas	December 25
Lincoln's Birthday	Monday closest to February 12
Independence Day	July 4
Memorial Day	Last Monday in May
Labor Day	First Monday in September
Thanksgiving	Last Thursday in November

4. If the 5th or 20th fall on a weekend and the preceding Friday is a holiday, payment will be made on the preceding Thursday.

PROBLEM 272

Under these conditions, anyone working for this firm would notice that the time between paydays varies widely. So we wish to find out the shortest elapsed time in days between paydays, and the longest such time, in the 50 years subsequent to January 1, 1980.

The problem is simple; that is, anyone can understand it. It is well defined. One method of solution is immediately apparent; namely, to obtain a calendar for the years 1980 through 2030, mark the paydays on it, and search for the longest and shortest periods between the paydays. Since there are some 1200 such periods to be searched, this approach would be tedious and error-prone.

On the other hand, it is something to do, and thus takes care of the ubiquitous wail "I don't even know how to get started." One would hardly expect to be able to look up in an engineering handbook for the topic "Pay days; elapsed time between." The problem requires a method of solution (whether by hand or by computer), and it is always an excellent idea to plunge in and do some of the problem solution by hand, however crudely. If nothing else, it will surely help to define the problem, and to gain some familiarity with its quirks. All problems have quirks.

Perhaps it isn't a computer problem, and one should start looking for alternative means of solution. The six criteria for intelligent use of a computer are also clearly defined. We might review them briefly:

1. The problem solution must be useful.
2. The problem must be defined.
3. A method of solution must be available.
4. The proposed solution must fit the available machine in both space and time.
5. There must be repetition in the solution.
6. The proposed solution must be cost/effective.

If you understand what is wanted and what is given (that is, the inputs and outputs), then the problem is defined. The instructions and data should fit any computer ever built, unless you propose to store fifty years of calendar data, and you shouldn't. There will be some 1200 pay periods to examine, which should satisfy anyone's definition of "repetition." And the payoff should be high; hand methods, or even punched card methods, would seem, offhand, to be awkward and inefficient--perhaps that calls for some faith at this point. Whatever it is that we propose to do by computer will be done very rapidly, so there is no problem about a time fit to the available machine. That leaves, if you've been keeping track, the matter of finding a method. We need an algorithm.

As we said, a good rule to follow is this: start to solve the problem the hard way, by hand and eye, tediously. When you get to the point where you find yourself complaining "Gee, I'll have to do that over 1195 more times," you are close to describing a computer solution. After all, the thing that computers do best of all is what they have just done--repetition is their forte. Your thinking up to that point constitutes most of the analysis of the problem and probably leads naturally to a logical flowchart, or its equivalent. The subsequent steps:

- Selecting some convenient language.
- Writing coded instructions in that language.
- Debugging the resulting program.
- Documenting the solution.
- Running production.

...are all more or less mechanical. The step called TESTING is still an art, and far from mechanical.

So the thing to do right away is to get a 1980 calendar and examine the first dozen cases by eye.

And right away we will find some salient features of the problem. The normal distance between paydays is 15.5 days; in our small sample we have found distances of 13 and 18 days. We have explored almost all the conditions, except for the one about a holiday occurring on a Friday next to a payday weekend.

We might notice that we will never be concerned with the days from the 7th to the 17th of any month, nor the days from the 22nd to the 2nd. We will be concerned with elapsed days numbering less than 13 or more than 18. We will probably need a mechanism to operate a calendar on a day-to-day basis. Of the 7 company holidays, only two thus seem to concern us; namely, July 4 and Labor Day.

The heart of our computer program will be a subroutine to simulate a calendar. The logic of such a subroutine is shown in a flowchart. We can number the days of the week from 1 to 7 for Sunday through Saturday, so that our calendar, when called, can go from:

1980 02 29 6
Y M D W

(February 29, 1980, Friday) to:

1980 03 1 7

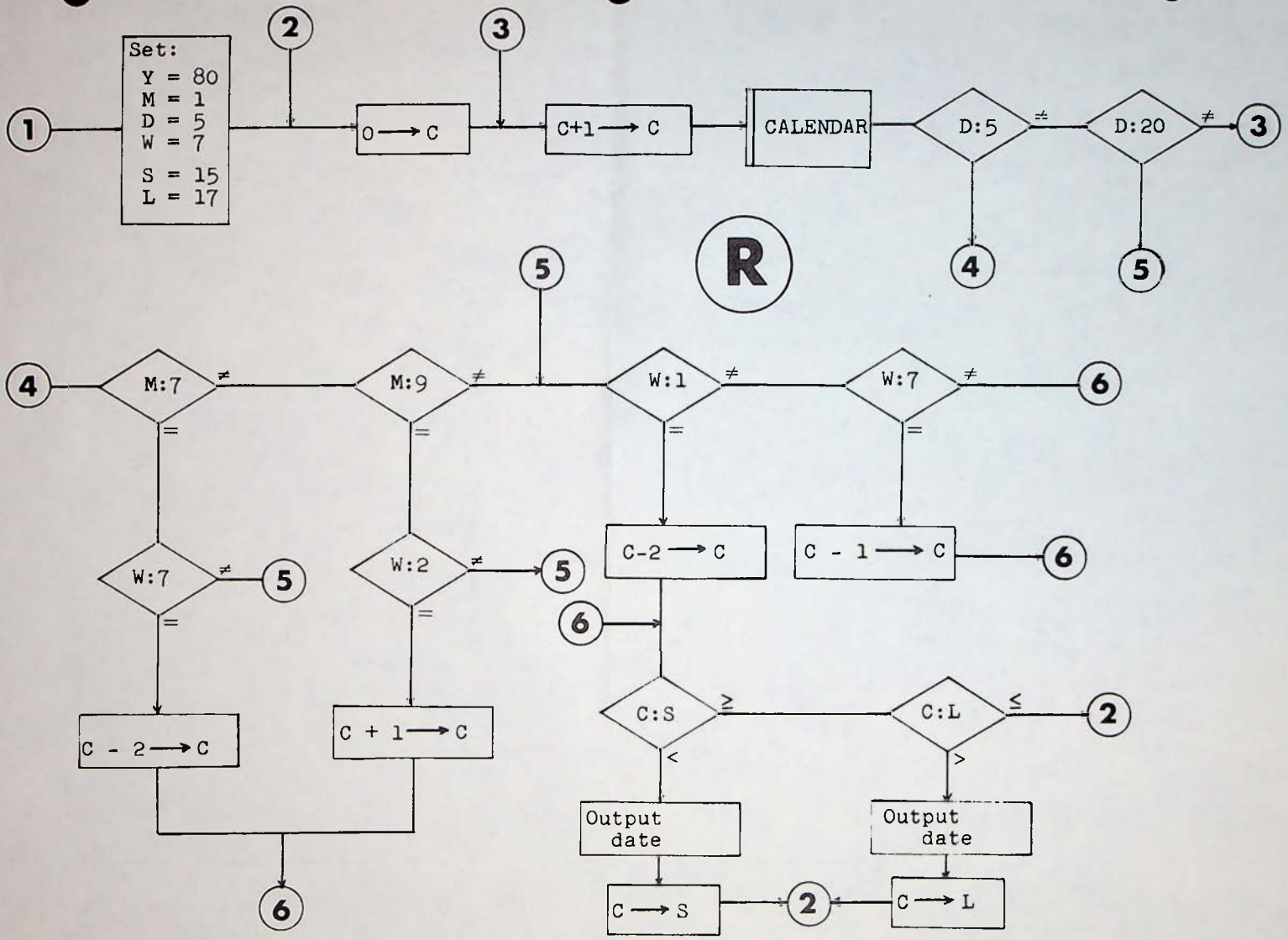
The calendar subroutine will be called 18262 times (the number of days in 50 years) unless we can devise a scheme to bypass those parts of the calendar that we know do not concern us.

Let's stop and consider that point. We could save perhaps as much as two-thirds of the production time on the machine if we could arrange to jump from day 7 to day 17 and from day 22 to day 2 without grinding those days out one by one. Is it worth it? There may be some time saved during debugging and testing, but that could not be of interest here. Suppose that the total time in production (that is, the time it takes to run through 50 years, searching for what we want) is on the order of 5 minutes, and we might be able to cut that down to, say, 1.5 minutes--at what cost? If it consumes a couple of hours of our time in order to implement the saving, we may be defeating ourselves. We may, in fact, increase the elapsed time to solution by a day or more, and that fact carries a cost, too.

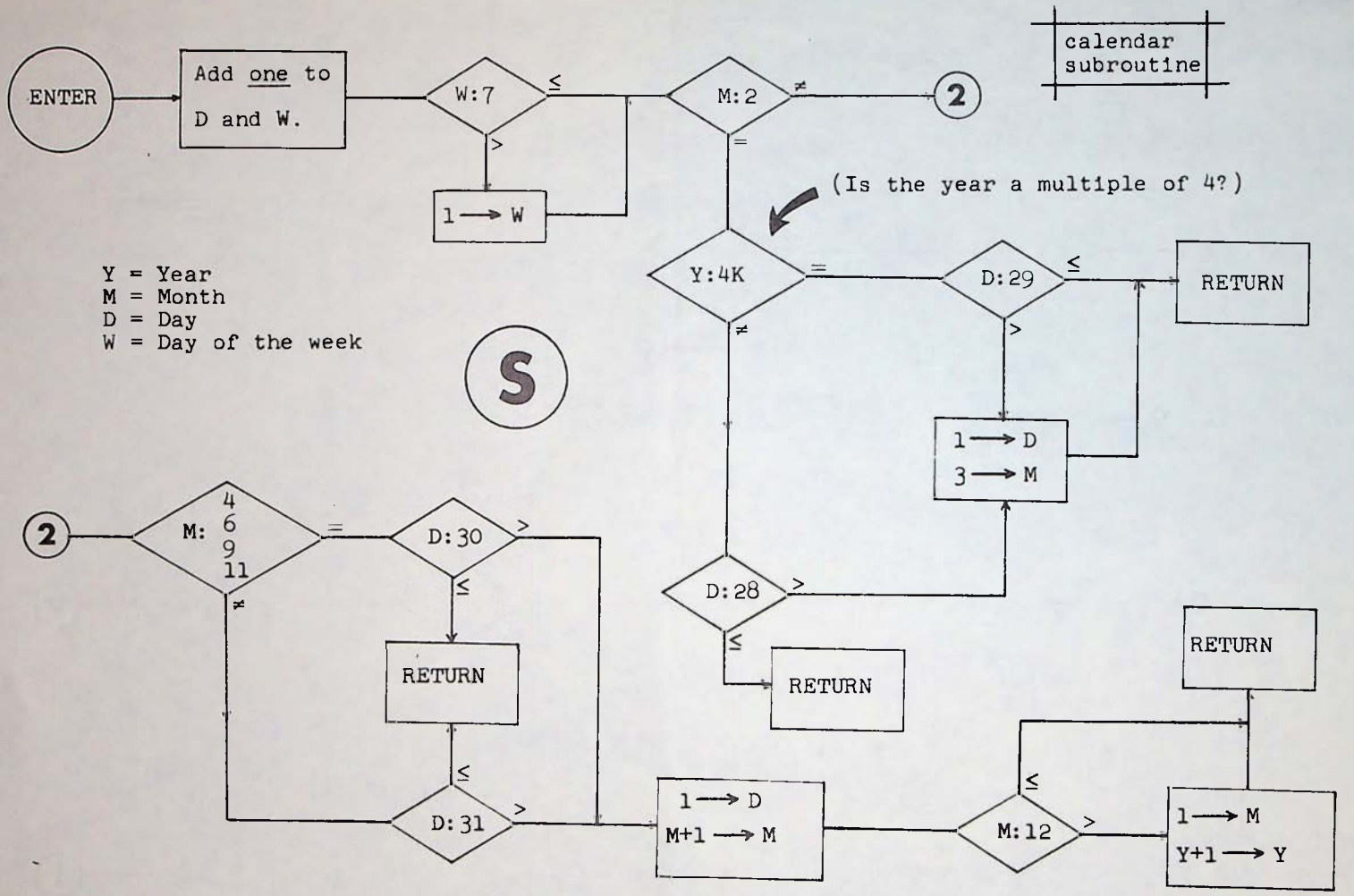
Decisions like the one above take considerable time to discuss in print. They are ever-present in computer problem solving, and the competent programmer learns to make them quickly; here is where experience and practice pay off. The experienced programmer would not hesitate over that decision for more than a few seconds, at most.

The flowcharts shown in Figures R and S represent a way to solve the Payday Problem. As is true for most computing problems, there are doubtless many other ways in which the problem could be tackled. Notice, however, that this is likely to be a one-shot deal. We may use the computer to examine 1200 pay periods and find, hopefully, the longest and shortest times between pay days; we are hardly likely to want to do that job again. Thus, a quick-and-dirty solution may be the best one. Which is not to say that a sloppy or incorrect solution is to be condoned--just one that may be somewhat inefficient.

In Flowchart R, Y = Year, M = Month, D = Day of the month, and W = Day of the week (Sunday through Saturday). S and L are for "Small" and "Large," representing starting values that we obtained from looking at a 1980 calendar. All the logic of Flowchart S is called as a subroutine, by the box labelled "Calendar" in Flowchart R. For the 50 years following 1980, any year divisible by 4 is a leap year.


Set:
Y = 80
M = 1
D = 5
W = 7
S = 15
L = 17
0 → C
C+1 → C
CALENDAR
D:5
D:20
R
M:7
M:9
W:1
W:7
W:7
W:2
C-2 → C
C - 1 → C
C - 2 → C
C + 1 → C
C:S
C:L
Output date
Output date
C → S
C → L

PC84-11


calendar subroutine
ENTER
Add one to D and W.
W:7
≤
>
1 → W
M:2
≠
2
=
(Is the year a multiple of 4?)
Y:4K
=
D:29
≤
RETURN
>
≠
1 → D
3 → M
Y = Year
M = Month
D = Day
W = Day of the week
S
2
M: 4 6 9 11
=
D:30
>
≤
RETURN
≠
D:28
>
≤
RETURN
RETURN
≤
D:31
>
1 → D
M+1 → M
≤
M:12
>
1 → M
Y+1 → Y

All this brings us to the point where we have a computing problem to consider. The analysis has been completed, up to the point of complete flowcharts. The choice of machine is probably constrained, for most of us; there is a machine available to us; few people enjoy the luxury of two or machines to pick from.

The only choice left, then, is that of language. This particular problem solution could be expressed in machine language; that is, absolute binary (or hexadecimal, as a convenient way of expressing binary). It could also be done conveniently in assembly language. But most people prefer, for many good reasons, to use some high level language, and almost any one will do (the outstanding exceptions being the specialized languages, like APT or SIMSCRIPT). The implementation of the logic of the flowcharts given here, in BASIC, would take less than 70 simple statements.

Incidentally, there will be a period of eleven days between paydays in early 1989.

19	(20)	21	22	23	24	25
26	27	28				

February 1989

			1	2	(3)	4
5	6	7	8	9	10	11

March 1989

There have been quite a few books published on the general topic of How To Solve Problems (the best of them by George Pólya; e.g., How To Solve It). All such books can be re-read profitably every year or so, since what they show is essentially "Here are some problems; I'll show you how I solved them."

Don't fall into that trap. We have shown here a way to solve the Payday problem. To be of any use, you must solve such problems (perhaps not this one; select one that interests you). Someone else's solution may be helpful if you're stuck along the way. But, as always, we are out to foster the notion that:

THE WAY TO LEARN COMPUTING IS TO COMPUTE

Book Review

Introduction to Computing--Structured Problem Solving Using WATFIV-S

by V. A. Dyck, J. D. Lawson, and J. A. Smith
(all of the University of Waterloo)

Reston Publishing Company, 1979, hard cover, 604 pages.

This is certainly the most comprehensive and wide-ranging text to bear the title "Introduction."

Although the title refers to WATFIV (a very high grade Fortran IV compiler), the book develops its programs in pseudocode, and an appendix shows the rules for transforming pseudocode into WATFIV. In the hands of thoroughly seasoned and experienced instructors, this approach would work, but in the hands of inexperienced instructors (that is, the majority of those teaching in Computer Science departments) the scheme would be chaotic. Not that this is in any way an adverse criticism of the book. The authors have written a text that suits them; that is, they have aimed extremely high. They should be encouraged. They can afford to aim high; they know what they're talking about.

The book includes not only the elements of introductory computing (but pointedly avoiding the use of flowcharts), but deals in detail with program structure, stepwise refinement, programming by modules, and top-down design. Much material that would normally be relegated to a semester course in Numerical Methods is included, as is material on sorting, statistics, and simulation.

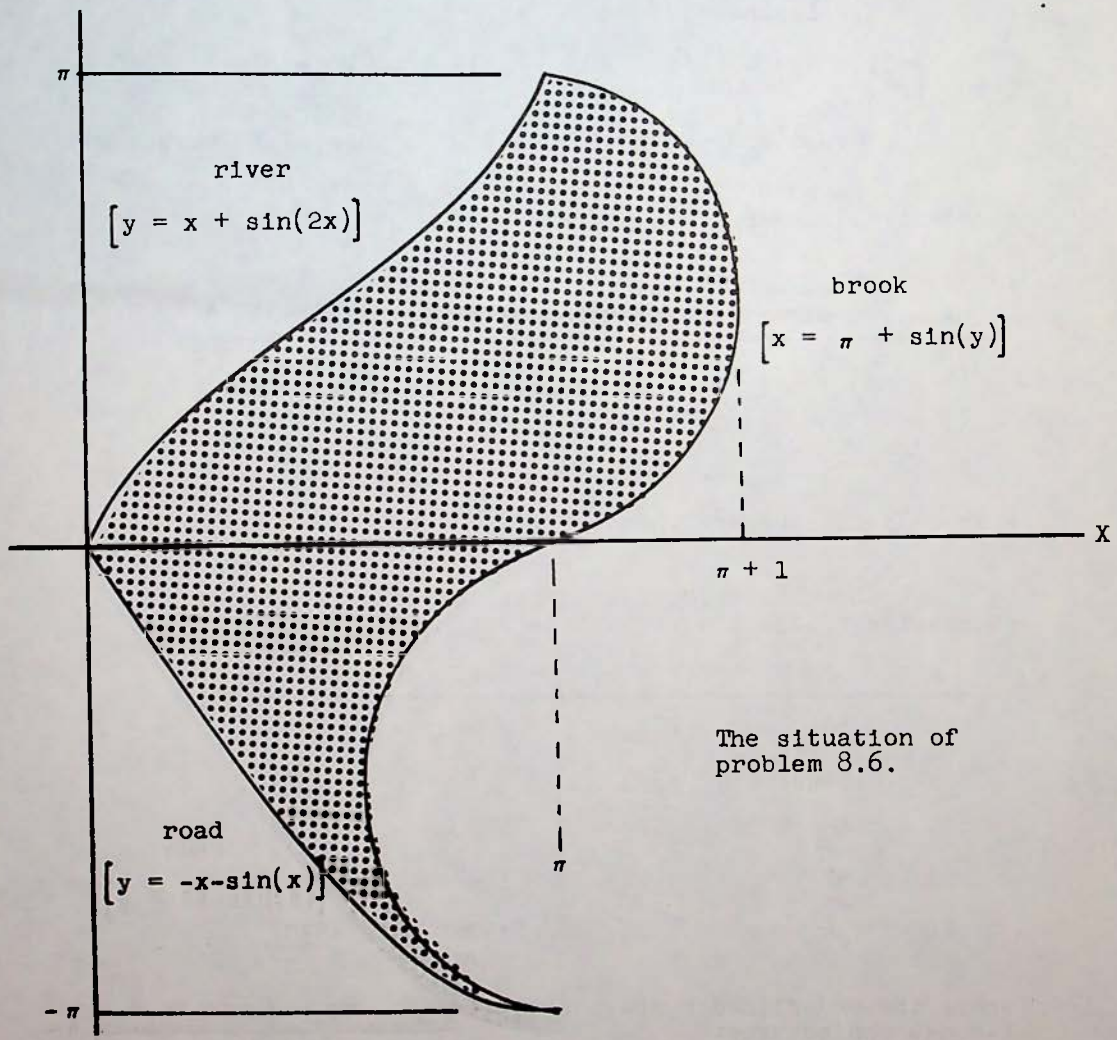
Interesting, challenging, and novel problems highlight the book (two of them are given after this review). The problems and exercises are carefully arranged to build on each other, reinforcing the principles being taught. Each Chapter includes up-to-date suitable references to outstanding source books.

There are 17 appendices (184 pages) including six complete files (e. g., National Hockey League Statistics) of data--a unique bonus. Every error message of WATFIV is listed, which is not only a handy reference, but which demonstrates the rich quality of the compiler.

The dust jacket refers to the book as "exciting." For once, a publisher has understated the case for a new book.

Following are two problems selected from the Dyck/ Lawson/Smith computing text.

8.6 A farmer has a very irregularly shaped farm bounded on one side by a winding brook, on a second side by a winding river, and on the third side by a curved road, as illustrated in the Figure. After considerable trial and error, he discovers the boundaries of his farm can be described by the equations indicated on the diagram. Develop a program to approximate the area of the farmer's farm.



The situation of problem 8.6.

4.13 It has been conjectured that the product of two consecutive numbers lies between two odd numbers of which at least one is prime.

(The material that follows is not from the textbook)

A moment's work furnishes two examples:

A	B	(PROD - 1)	PROD	(PROD + 1)	
6	7	41	42	43	**P**
7	8	55	56	57	**Q**

where A and B are the consecutive integers. Case P supports the conjecture; case Q destroys it. Indeed, for the first 100 values of A, 35 are like case Q. Thus, for A = 1, 2, 3,...,99 there are 65 P cases, and 35 Q cases; the ratio of Q/N is thus .35.

So we have the following problems to be explored by computer:

1. How does that ratio (of Q/N) change as N increases?

2. Up to A = 1000, find the longest string of successive Q cases, such as this string of 3:

A	B	(P - 1)	P	(P + 1)
72	73	5255	5256	5257
73	74	5401	5402	5403
74	75	5549	5550	5551

where all the numbers in columns (P-1) and (P+1) are composite.

3. Up to A = 1000, find the longest string of successive P cases, such as this string of 5:

A	B	(P - 1)	P	(P + 1)
53	54	2861	2862	2863
54	55	2969	2970	2971
55	56	3079	3080	3081
56	57	3191	3192	3193
57	58	3305	3306	3307

where the underlined numbers are prime. We seem to have two new conjectures:

1. There is a longest string of successive P cases.
2. Strings of successive Q cases will continuously get longer as N increases.

Book Review...

The Information Age

by William S. Davis and Allison McCormack

Addison-Wesley 1979, 427 pages hard cover $13.95.

This book arrived for review early in 1979, and was discarded as not worth reviewing.

Then the November 1979 issue of Datamation carried a review, by Phil Dorn, in which he says

> "This textbook comes very close to fulfilling the need for a single book for an 'Introduction to Computers' course."

With an accolade like that, and from as eminent an expert as Dorn, the book warrants a second look.

Addison-Wesley would do well to promote the book with Dorn's words. The book just doesn't live up to its billing.

Try some facts, first. No, card readers don't wrap the cards around a cylinder to read them--have the authors never seen a card reader opened up? And photo-electric readers don't sense all 960 hole positions at once; they sense either 12 holes or 80 holes, depending on which direction the card is read. Does it make any difference whether or not such things are correct? Should it not be the first requirement of a textbook that the writers tell no lies? And if they are off in small factual things, does it not imply that they do not know what they are talking about?

Next, a text should present its material at a proper level. Presumably, a text for an "introduction to computers" course should be at the level of, say, a typical junior college student. So let's say that you want this student to comprehend the two basic building blocks of programming; namely, loops and subroutines.

Davis and McCormack dismiss loops with three para-graphs, but not the building-block kind of loop; they know only the go-back-and-read-another-card type of loop. Even when explaining BASIC, they omit the FØR...THEN type of loop, and in explaining Fortran, there is no mention of its most powerful construct, the DØ loop.

Well, how about the other basic tool, the closed (linked) subprogram or subroutine? Their treatment of this topic is even briefer; here it is, complete:

> "Programmers often make use of subprograms and other prewritten modules that must be added to the object code by another special program called a linkage editor."

which certainly clears up that topic.

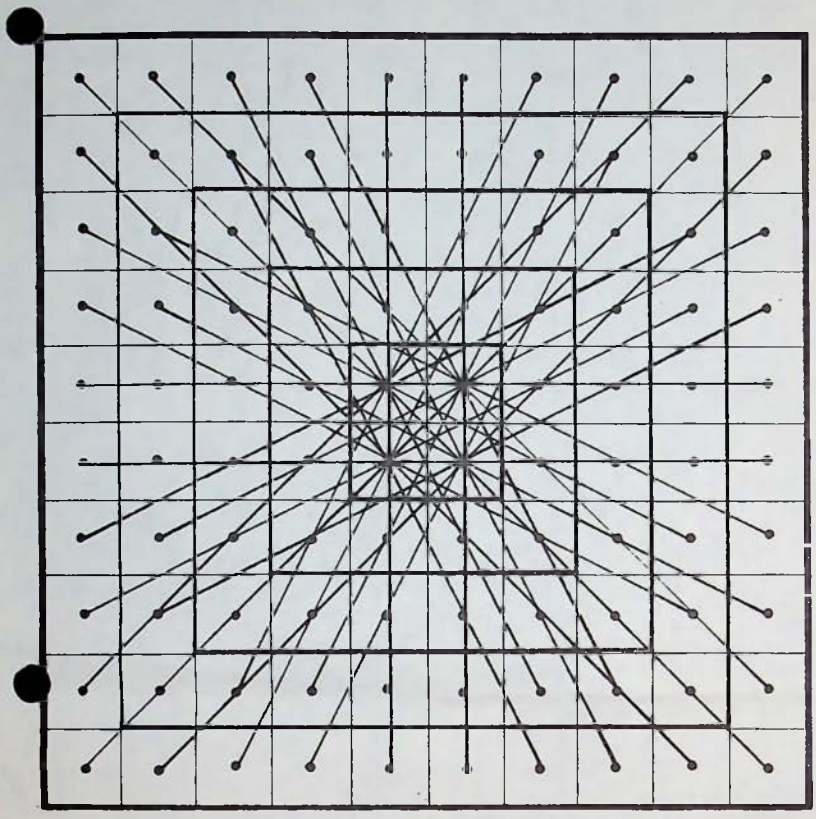
Just what book did Mr. Dorn read? The history section of this book has only a passing reference to von Neumann, and that with his name misspelled. Much credit is given to Atanasoff as the inventor of the computer, but by the authors' own definition of "computer" in their glossary, they are being inconsistent. They do, however, perpetuate the nonsense that direct access and random access are the same thing.

The two authors are much given to exclamation points (even junior college students must resent being talked down to), but then it probably is a surprise to them to them to find that 32 bits can hold numbers in excess of two billion! or that computer controlled printers can produce 20 to 30 characters per second! (Emphasis theirs.)

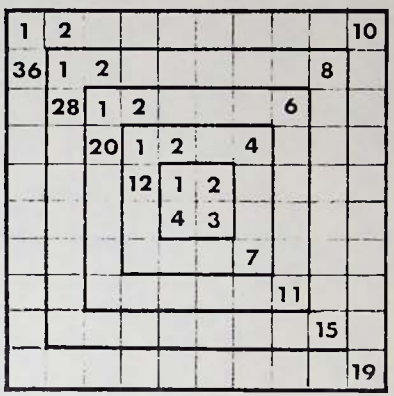
Some 40% of the book is devoted to the computer-and-society part of a computer appreciation course. The bulk of it is the Gee Whiz description of computing that appeals to young science-fiction addicts and, in fact, frequent references are to science fiction. All the apocryphal computer stories are repeated as fact. Every myth ever generated is reported as gospel.

It would be useful and nice if there were a single outstanding text for an "Introduction to Computers" course. But this one isn't it.

Problem Solution



The Five Square Rings



PC84-19

Problem 266 called for selecting one cell at random in each of five rings (numbered 1 to 5, outer to inner), and determining the probability that the five cells would lie on a straight line. The problem was attacked by two Computer Science students at California State University, Northridge: Janis Rangno and Dean Stoutland.

It was observed that the number of sets of five that lie on a straight line is small; they can be pictured and tabulated, as shown here. There can thus be an analytic solution:

$$\frac{6(2^5) + 4(2^4) + 8(2)}{(36)(28)(20)(12)(4)} = .00028108$$

by comparing the sets of five that are collinear, with all possible sets of five. A Monte Carlo attack on the same problem should confirm the analytic probability.

Unfortunately, with such a small probability, making 10,000 trials would furnish barely one significant digit of the result. The two students collectively made three runs of 10,000 trials each, obtaining results of 4, 3, and 2 successes respectively, corresponding to an empirical result of .0003.

Outer ring	Ring 2	Ring 3	Ring 4	Inner ring
1 or 19	1 or 15	1 or 11	1 or 7	1 or 3
2 or 18	2 or 14	2 or 10	2 or 6	2
3	17	2	8	1
4	16	3	7	2
5 or 24	4 or 19	3 or 14	2 or 9	1 or 4
6 or 23	5 or 18	4 or 13	3 or 8	2 or 3
7	21	4	10	1
8	20	5	9	2
9 or 29	7 or 23	5 or 17	3 or 11	1
10 or 28	8 or 22	6 or 16	4 or 10	2 or 4
11 or 27	9 or 21	7 or 15	5 or 9	3
12	24	7	11	2
13	23	8	10	3
14 or 33	11 or 26	8 or 19	5 or 12	2 or 1
15 or 32	12 or 25	9 or 18	6 or 11	3 or 4
16	28	9	1	2
17	27	10	12	3
20 or 36	16 or 28	12 or 20	8 or 12	4
21	3	12	2	3
22	2	13	1	4
25	7	14	4	3
26	6	15	3	4
30	10	17	5	4
31	9	18	4	1
34	14	19	7	4
35	13	20	6	1

A tabulation of all possible successful cases in the Five Square Rings problem. Each line indicates the cell numbers of five cells, one from each ring, that do lie on a straight line.